COMPLETE LISTING OF THE CLAIMS

This listing of claims will replace all prior versions and listings of claims in the application. Claims 3 and 16 have been amended and new claims 41 and 42 have been herein.

1. (Original) A system for facilitating an interface dispatch, comprising:

a pre-execution engine adapted to load source code and allocate a block of memory in the form of a vector for creating an interface map that includes a plurality of slots for referencing interface virtual tables that correspond to an implementation of an interface in a class type; and

an interface index component adapted to assign index numbers to interfaces as the interfaces are loaded by the pre-execution engine, the pre-execution engine being adapted to determine a row structure for a class type for a plurality of class types based on a class type and interfaces implemented by the class type using the indices assigned to the interfaces, the pre-execution engine associating indices with empty slots in the interface map based on the configuration of the row structure and storing references to the interface virtual tables in the empty slots when enough empty slots are found for the respective row structure.

2. (Original) The system of claim 1, the pre-execution engine being adapted to store row structures utilizing a comb-vector technique.

3. (Currently Amended) The system of claim [[1]] 2, the comb-vector technique comprising conceptually sliding row structures corresponding to class types within the interface map until each interface index entry hits an empty slot.

4. (Original) The system of claim 1, the pre-execution engine being adapted to assign a row start location upon finding empty slots for each interface index entry, the row start location providing a start location for the row structure and the interface index numbers providing offsets from the row start location.

5. (Original) The system of claim 1, the row start location for one class type being the same as a row start location for another class type as long as collisions between row structures are avoided.

6. (Original) The system of claim 1, the pre-execution engine being one of a loader and a linker

7. (Original) The system of claim 1, the pre-execution engine being a compiler.

8. (Original) The system of claim 7, the compiler being a JIT compiler.

9. (Original) The system of claim 1, the indices being assigned sequentially.

10. (Original) The system of claim 1, the pre-execution engine determining a row start location in the interface map for each row structure, such that a reference to an interface implemented by a class type in a slot location can be accessed using the row start location and the index number corresponding to the desired interface.

11. (Original) The system of claim 10, a reference to an interface implemented by a class type in a slot location can be accessed by adding the row start location to the index number corresponding to the desired interface.

12. (Original) The system of claim 11, the row start location being stored in a method table corresponding to a class type.

13. (Original) The system of claim 1, the slot size being 32-bits and the slot being accessed by multiplying an index number by four and adding it to a row start location.

14. (Original) The system of claim 1, the pre-execution engine being adapted to dynamically create additional interface maps as the interface maps are filled.

15. (Original) The system of claim 1, the pre-execution engine being adapted to create a special map for all COM classes and COM interfaces.

16. (Currently Amended) The system of claim 1, further comprising an execution engine adapted to receive a method call of an interface for a specific class type and ~~accessed~~ access the interface virtual table corresponding to the interface and class type utilizing the interface map.

17. (Original) The system of claim 16, the execution engine locating a slot location having a reference to an interface virtual table corresponding to an interface implemented by a class type using a row start location and the index number corresponding to the desired interface.

18. (Original) A method of creating a reference map to interface virtual tables, comprising:

allocating a vector in memory of a predetermined size having a plurality of slots for creating an interface map for storing references to interface virtual tables;

loading source code having a plurality of classes and plurality of interfaces;

assigning indices to the plurality of interfaces;

determining a row structure for a class type and interfaces implemented by the class type utilizing the indices; and

storing references to interface virtual tables corresponding to the row structure in the plurality of slots utilizing a comb-vector technique.

19. (Original) The method of claim 18, further comprising repeating the steps of determining a row structure and storing references for each class type that implements interfaces in the source code.

20. (Original) The method of claim 18, further comprising allocating a second vector in memory of a predetermined size upon the interface map becoming full.

21. (Original) The method of claim 18, further comprising creating a second interface map for storing references to COM interfaces.

22. (Original) The method of claim 18, the comb-vector technique comprising conceptually sliding row structures corresponding to class types within the vector until each interface index entry hits an empty slot.

23. (Original) The method of claim 18, further comprising assigning a row start location upon finding empty slots for each interface index entry, the row start location providing a start location for the row structure and the interface index numbers providing offsets from the row start location.

24. (Original) The method of claim 23, further comprising storing the row start location for a class type in a method table corresponding to the class type.

25. (Original) The method of claim 23, the row start location for one class type being the same as a row start location for another class type as long as collisions between row structures are avoided.

26. (Original) The method of claim 23, further comprising accessing a reference in a slot location of the interface map by using the row start location and the index number corresponding to the desired interface and class type.

27. (Original) The method of claim 23, further comprising performing a casting of a class instance to an interface utilizing the interface map.

28. (Original) A computer-readable medium having computer executable components comprising:

a pre-execution engine component adapted to load source code having a plurality of classes implementing interfaces, the pre-execution engine assigning indices to the interfaces as the interfaces are loaded;

an interface map component that includes a plurality of slots for referencing interface virtual tables that correspond to an implementation of an interface in a class type, the pre-execution engine being adapted to determine a row structure for a class type based on a class type and interfaces implemented by the class type using the indices assigned to the interfaces, the pre-execution engine associating indices with empty slots in the interface map based on the configuration of the row structure and storing references to the interface virtual tables in the empty slots when enough empty slots are found for a respective row structure.

29. (Original) The computer readable medium of claim 28, further comprising an execution engine component adapted to receive and execute code and access the interface virtual table corresponding to the interface and class type utilizing the interface map in response to a method call for a specific interface type implemented in a class instance of a specific class type.

30. (Original) The computer readable medium of claim 29, the execution engine component accessing the interface virtual table corresponding to the interface and class type utilizing the interface map by using a row start location corresponding to a class type and adding the index number associated with the desired interface to the row start number to access the reference to the interface virtual table.

31. (Original) The computer readable medium of claim 28, the pre-execution engine being adapted to store row structures utilizing a comb-vector technique comprising conceptually sliding row structures corresponding to class types within the interface map until each interface index entry hits an empty slot.

32. (Original) The computer readable medium of claim 28, the pre-execution engine component being adapted to assign a row start location upon finding empty slots for each interface index entry, the row start location providing a start location for the row structure and the interface index numbers providing offsets from the row start location, the row start location being stored in a method table corresponding to the class type associated with the row structure.

33. (Original) The computer readable medium of claim 28, the indices being assigned sequentially.

34. (Original) The computer readable medium of claim 28, the pre-execution engine component being adapted to create the interface map.

35. (Original) The computer readable medium of claim 34, the pre-execution engine component being adapted to dynamically create additional interface maps as the interface maps are filled.

36. (Original) The computer readable medium of claim 34, the pre-execution engine component being adapted to create a special map for all COM classes and COM interfaces.

37. (Original) The computer readable medium of claim 28, the interface map being a vector.

38. (Original) A system for facilitating an interface dispatch by providing an interface map containing references to interface virtual tables, comprising:

means for allocating a block of memory for creating an interface map with a plurality of slots;

means for loading source code having a plurality of class types implementing interfaces;

means for assigning indices to interfaces as the interfaces are loaded;

means for determining a row structure based on the class type and the interfaces implemented by the class type employing the indices; and

means for storing references to interface virtual tables based on the row structure of each class type using a comb-vector technique.

39. (Original) The system of claim 38, the comb-vector technique comprising conceptually sliding row structures corresponding to class types within the interface map until each interface index entry hits an empty slot.

40. (Original) The system of claim 38, further comprising means accessing the references in the slots, the means for accessing the references using a row start location corresponding to a class type and adding the index number associated with the desired interface to the row start number to access the reference to the interface virtual table.

41. (New) An interface dispatch system, comprising:

a pre-execution engine that loads code and allocates memory for an interface map in the form of a one-dimensional array that includes a plurality of memory slots that reference interface virtual tables that provide references to implementations of one or more class interfaces; and

an interface index component to assign index numbers to interfaces as they are loaded by the pre-execution engine, the pre-execution engine determines a row structure defining spatial relations amongst interfaces of the same class based on the class, interfaces implemented by the class and the indices assigned to the interfaces, associates indices with

empty slots in the interface map based on the row structure, and stores references to interface virtual tables in the empty slots if the map can accommodate the row structure, otherwise a new interface virtual table is created.

42. (New) A method of accessing interface implementation methods comprising:

receiving a method call associated with a particular interface;

determining the class that implements the interface;

retrieving an interface map row start value from a method table associated with the class that implements the interface, the row start identifying the first slot of a row structure, which defines spatial storage relations amongst a plurality interfaces of the same class by indicia, for the particular class;

retrieving an index offset associated with the interface;

adding the offset to the row start value to locate the slot in the row structure associated with the particular interface;

retrieving the address for the appropriate interface virtual table from the located slot; and

locating and executing the received method via the interface virtual table.